View metadata, citation and similar papers at core.ac.uk
brought to you by CORE
provided by UCT Computer Science Research Document Archive

An Application of Tetrahedrisation to From-point Visibility

Technical Report CS04-12-00

Department of Computer Science
University of Cape Town
B. Miszka, G. Ryan, J. Gain and C. Hultquist

ABSTRACT

We propose a method for tetrahedrizing a polyhedral volume containing polyhedral holes. We use the resulting tetrahedrization to produce from-point visibility algorithms that can be either exact, conservative or aggressive. We also discuss the applications of such from-point and from-region visibility techniques, including their use in lighting and in simulating vision of artificial intelligence agents.

1. INTRODUCTION

In typical 3D scenes most primitives are not visible from a given viewpoint. Thus it would be desirable to determine which primitives are invisible during rendering so as to save rendering calculations. Algorithms that eliminate invisible primitives are known as visibility culling algorithms.

We propose tetrahedrising the viewpoint space (filling the space between objects with pyramid-like structures). It is assumed that the 3D scene is defined by a number of triangular meshes. The tetrahedrisation will be performed such that the face of each tetrahedron is either the face of a triangle that is part of one of the triangular meshes that make up the scene, or co-incident with the face of an adjacent tetrahedron. This will occur as a pre-procces. The resulting tetrahedrisation will then be used in conjunction with a from-point visibility algorithm that can be tuned to be exact, conservative or aggressive. The tetrahedrisation could also be used as the basis for from-region visibility techniques.

2. BACKGROUND

The approach is divided into two sections, namely tetrahedrisation and the from-point visibility culling algorithm. Thus the background and related work will be discussed separately for each of the two sections.

2.1 Tetrahedrisation

There is a large amount of literature describing tetrahedrisation, mostly regarding its application in mesh generation. Bern [3] provides a detailed server of mesh generation techniques and Owen [19] provides a survey focused on unstructed mesh generation. Bern and Eppstein [4] provide a more detailed discussion on mesh generation that focuses on approaches using triangulation. They address many aspects of triangulation in both two dimensions and three dimensions as well as specifying open problems.

In structured mesh generation generation, interiour vertices are topologically alike whereas in unstructured mesh, vertices can have arbitrarily varying local neighbourhoods (Bern [3]). It is far simpler to respect object boundaries when using unstructured mesh generation and thus it is more applicable to this project.

There are two sub-goals of the tetrahedrisation component. The first is to attempt to minimise the number of tetrahedral and the second is to produce good quality tetrahedra. It is desirable to produce tetrahedra that are of high quality under all measures because in other applications of tetrahedrisation, particularly those involving finite element analysis, quality is important in minimising numerical errors [7].

There are several quality measures for tetrahedra. These include aspect ratio, minimum dihedral angle, maximum dihedral angle and radius-edge ratio [12].

Regular tetrahedra are of the highest quality under all measures, thus it would be desirable to tile the viewpoint space with regular tetrahedra. This is impossible however, as shown by Eppstein et al [12]. Tetrahedral shapes that can be used to tile three-dimensional space do exist. However, none of these tetrahedra are acute, though some are nonobtuse [12].

Although tetrahedrisation is effectively the triangulation problem in three dimensions, it is more complicated than the two dimensional case. This is due to the fact that many properties of two-dimensional triangulation break down in three dimensions. Different triangulations of the same input may contain different numbers of tetrahedra. A generalisation of Euler's formula shows that the tetrahedrisation of an n-vertex polygon has at most $\binom{n-2}{2}$ tetrahedra. However, tetrahedrisation can often be accomplished using far fewer tetrahedra, especially if the polygon

is strictly convex where tetrahedrisations that are linear in the number of polygon vertices can be performed (Bern and Eppstein) [4]

An additional complication that arises in three dimensions is that not all polyhedra can be tetrahedrised without the addition of Steiner points. Schönhardt provided an example of a polygon of this type [22].

These two issues then raise the question as to the number of tetrahedra and the number of Steiner points required to tetrahedrise an arbitrary polyhedron. Bern and Eppstein show that an polyhedron can be tetrahedrised with $O(n^2)$ tetrahedra and $O(n^2)$ Steiner points. These two measures form an upper bound and convex polyhedra often require fewer Steiner points and fewer tetrahedra (as mentioned earlier).

Unfortunately, it has been shown that testing to determine whether or not Steiner points are required to tetrahedrise a polyhedron is NP-complete. (Rupert and Seidel) [21]. They also prove that for any k, it is NP-hard to test whether or not k Steiner points will suffice.

Apart for their being required in the tetrahedrisation of certain polygons, Steiner points may also be used to improve the quality of the tetrahedra obtained. In fact, Steiner points can also reduce the complexity of the tetrahedrisation and Bern, Eppstein and Gilbert prove that their technique for two dimensional triangulation (based on quadtrees) can be extended to any fixed dimension, giving a triangulation of that is of size O(n), where n is the number of input vertices [5]. However, they do not provide the extension of their algorithm. It is still an open problem to triangulate an arbitrary convex polygon with the minimum number of tetrahedral in polynomial time [3].

Acute triangulation in three dimensions has not been addressed literature until recently by Eppstein et al [12]. Eppstein et al [12] provide a method for the acute triangulation of a slab of three-dimensional space. However, although the tiling fits between two parallel planes, it has “dimples” on the outer surfaces, making it unsuitable for tetrahedrisation of arbitrary polyhedra as it stands. They state that finding an acute tetrahedrisation of a given shape, such as a cube is still an open problem. Additionally, acuteness alone is not a guarantee of quality of tetrahedra. Three types of bad quality tetrahedra can have all their interior angles acute [12].

Typical approaches to tetrahedrisation are Delaunay based or octree based. Delaunay based methods use the three-dimensional extension of the two dimensional Delaunay triangulation. Octree based methods recursively divide space into cubes and then triangulate these cubes while taking into account the input polyhedra. The Delaunay triangulation in two dimensions has the property of minimising maximum radius over all triangles' circumcircles. The Delaunay triangulation can be extended to three dimensions; however certain properties of the triangulation that hold in two dimensions, such as acuteness, no longer hold in three dimensions [19]. However, the Delaunay triangulation does maintain useful properties in three dimensions such as it minimizing the maximum radius of a min-containment sphere (Rajan) [20]. Delaunay based approaches do not eliminate all types of poor quality tetrehedra, such as, slivers may still be present and the mesh must be improved as a post process (S. Cheng, K. Dey, H. Edelsbrunner, M. Facello, S. Teng [8]).

There is no analogue of the constrained Delaunay triangulation (CDT) in three dimensions. Typical approaches thus find the Delaunay triangulation of the given object's point-set, forming the convex hull of those points. The object boundary is then recovered. Cavalcanti and Mello [7] provide the outline details for an approach to three-dimensional constrained Delaunay triangulation suitable for industrial applications. Their main emphasis is on recovery of constraining faces and edges while minimising the use of geometric operations such as intersections.

Du and Wang [11] also propose a constrained boundary recovery technique. In fact they build on their previous technique that performs a conforming boundary recovery. An attempt is made to minimise the number of Steiner points required. The technique does rely on heuristics unlike the approach used by Cavalcanti and Mello that uses an heuristic for Steiner point insertion for face recovery.

Mitchell and Vavasis [16] extend the technique for two dimensional triangulation with bounded aspect ratio, described by Eppstein et al [5], to tetrahedrisation in three-dimensions. Their approach produces tetrahedra that have aspect ratio within a constant factor of optimal. The number of tetrahedra is also within a constant factor optimal for a bounded aspect ratio tetrahedrisation.

2.2 From-point Visibility Culling

The area of visibility in computer graphics has been around since the early days of the field (In March 1974, Sutherland, Sproull and Schumaker [23] surveyed ten hidden surface algorithms available at the time). There have been many algorithms designed to solve this problem. In general, these can be divided into two categories; from-point and from-region.

The from-point algorithms perform just as the name implies, determining what is visible from a specified viewpoint. The from-region techniques are aimed at determining what is visible from all points in a certain area or region in a scene. From-region techniques also occur largely as a pre-process (occurring before the actual running of programs which would use the visibility information), while from-point approaches are more run-time oriented.

The various approaches for from-point methods may also be divided according to their accuracy. Nirenstein [18] provides the following classifications:

- *Exact:* such an algorithm returns exactly that which is visible from a point.

- *Conservative:* provides an overestimate of what is visible from a point. This means primitives that should be invisible are marked as visible.
- *Aggressive:* provides only that which should be drawn but may occasionally falsely mark some visible primitives as invisible.
- *Approximate:* suffers from both false visibility and invisibility errors.

The algorithms may also be categorised according to the way in which they operate. The large majority of techniques can be grouped into two categories:

- *Object precision:* Visibility computations are performed using the raw models or information representing an object
- *Image precision:* operate on discrete representations of objects broken into fragments during the rasterisation (conversion to pixels) process.

Our proposed algorithm can be classified as an exact method, and as an object precision method because it works with the 3-D representations of the tetrahedrisation.

A brief examination of some object precision and image precision methods follows. The methods below are considered to be conservative approaches to visibility (unless stated otherwise).

3.2.1 Object precision methods

- *Cells and portals*
  The scene under consideration is divided into separate cells, which are joined by portals, and moving from one cell to another can only be accomplished by moving through portals (it is useful to think of cells as rooms, and portals as the doors joining them). It is a conservative method.
  To begin with, the visible primitives in the current cell are determined, and then rays cast into other cells to determine what is visible in them. It is useful to note, that cell contents will only need to be considered if one of the portals to that cell is visible (since it is only possible to move into cells through portals). If all portals to a certain cell are not visible, then anything contained in that cell is also not visible.
- *Large convex occluders*

  Coorg and Teller [10] present a method whereby they characterise the occlusion of a single convex occluder using the separating and supporting planes between them, and the position of the viewer with respect to those planes. (see Figure 1 for a graphical view of such a situation). The basic idea is that if an observer is between the supporting planes and behind an object (the *occluder*), then it is not possible to see the other object (the *occludee*). If an observer is between separating and supporting planes, then it is possible to see part of an object (the object is only partially occluded). Finally, if the observer is outside both separating and supporting lines, then it is possible to completely see the occludee object.
- *Culling using shadow frustra*
  This idea was proposed by Hudson *et al.* [14], and makes use of the fact that if an object lies completely within the "shadow" of another object (with respect to the view-point) then that object is not visible. Care needs to be taken to accommodate partially "shadowing" of objects.
- *BSP (Binary Space Partitioning) tree culling*
  The shadow-frustra culling of Hudson *et al.* [14] can be improved by using BSP trees. Bittner, Havran, and Slavik [6] combine the shadow frustra of various occluders into an *occlusion tree,* which is then used to compare against the scene hierarchy.

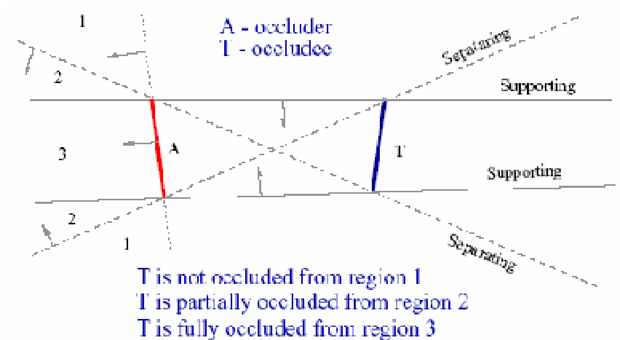

Figure 1: An example of visibility determination using convex occluders (figure was obtained from Cohen-Or *et al.* [9]).

Weiler and Atherton [25] also present a noteworthy method. They provided both a system for hidden surface removal in scenes as well as a polygon clipping technique. The system primarily uses depth and area sorting to determine the ordering and visibility of polygons. The polygons are then clipped against closer polygons to determine their visible portions (this is an exact method).

3.2.2 Image precision methods:

- *Ray casting*
  This involves casting a ray from the "eye" through every pixel of the screen. The pixel value is determined by the intersection with the closest object. This can be expensive because of tests against every object in the scene, but when a back to front ordering of rendering is used it performs natural occlusion culling (this can be fast as demonstrated in Bala, Dorsey, and Teller [1]).
- *Hierarchical Z-buffer (HZB)*
  An extension to the Z-buffer proposed by Greene, Kass, and Miller [13], this method makes use of octrees (a way of subdividing/partitioning 3-Dimensional space) and a Z-pyramid (layered buffer with different resolutions at each level). The scene is first arranged into an octree, and traversed top-down, front to back, with each node being tested for occlusion. If a node is found to be occluded then it is skipped, otherwise its children are tested. The Z-pyramid is updated during scan-conversion of the primitives, and any z-buffer changes are propagated up the pyramid. To determine

visibility of a node, each face is hierarchically tested against the z-pyramid.

- *Hierarchical occlusion buffer*
  Proposed by Zhang *et al.* [26], it is similar to the HZB, the occlusion is arranged hierarchically into a structure known as the *Hierarchical Occlusion Map* (HOM), and the bounding volume hierarchy of the scene is tested against it. However, the HOM stores only opacity information, with distance values being kept elsewhere. This may also be an aggressive method.
- *Approximate Volumetric Visibility*
  This is an approach that does not use geometric visibility computations, but instead relies on the creation of volumes/regions that have certain properties. Klosowski and Silva [15] developed such an approach in their *prioritised layered projection* (PLP) system. It estimates the visible primitives in a scene, using a pre-assigned probability that a cell is visible from a given viewpoint.

There is also a method described by Bartz, Meiner, and Httner [2], which uses OpenGL calls to perform the testing. Hardware implementations of culling approaches have also been explored, with some being incorporated into modern graphics hardware (ATI's Hyper-Z technology [17], for example).

4. APPROACH

The tetrahedrisation program reads in the coordinates and facets from a VRML 2.0 file. From this it constructs indexed lists of coordinates, edges and facets. The tetrahedrisation is computed and written out in a simple format to be read in by the visibility culling algorithm.

4.1 Tetrahedrisation

The approach used is that proposed by Mitchell and Vavasis [16]. The input is given as a connected polyhedral region P in R^3. P is specified by mutually linked lists of vertices, edges and facets. The main data structure used by the algorithm is an octree. An octree is a rooted tree, where each node is either a leaf node or has exactly eight children. Each node of the octree is referred to as a box. Each box represents a polyhedral region called its embedding that is denoted I(b). During octree generation, the embedding of each box is a cube. In later stages of the algorithm, when boxes are warped and triangulated, their shape is modified.

The embeddings of the eight children of a box are obtained by dividing the embedding of the box into eight equal cubes. This is accomplished by dividing the embedding in half in each of the three dimensions. All boxes that are not leaf nodes are referred to as split. Thus the processing of dividing a box into its eight children is called splitting.

There are a number of terms that must be defined in order to explain the algorithm.

Duplicate: When the intersection of a given box with P, denoted by Pn I(b), is found to have more than one connected component, the box is duplicated into the original box and a number of new nodes called duplicate boxes. Each cube represents the same region in R^3, but each is associated with one connected component of Pn I(b). P?b is used to denote the component of Pn I(b) associated with a given box. Note that if P?b is non-convex, a child box of b may have more than one component, even though P?b does not. Thus, whenever any box is split, each of its child boxes must be examined and duplicate boxes created for each child where necessary.

Extended Box: Given a box b, its extended box, denoted by ex(b), is defined such that I(ex(b)) is a cube concentric with I(b) but expanded by a factor of five in each dimension. P?ex(b) is used to denote the component of Pn I(ex(b)) that contains P?b. ex(b) Is not stored explicitly in the octree but the P faces it contains can be deduced from boxes higher up the tree than b.

Adjacent: Two boxes are defined as neighbours if their embeddings intersect non-trivially and they are not duplicates of one another. Two boxes that are neighbours are said to be adjacent if there is a point of P common to both of them. Two boxes are balance-adjacent if they are neighbours and there is a point of P common to one of the boxes and the extended box of the other.

Balance Condition: For the purposes of this algorithm, the size of a box is defined to be the length of an edge of the box. The balance condition that is maintained is as follows: no box may be balance adjacent to another box that is more than twice its size. Thus whenever a box is split, the boxes balance adjacent to its children must be examined and split if necessary to maintain the balance condition. These splits may necessitate the splitting of other boxes to propagate the balance condition. Note that certain boxes are protected during the octree generation and thus these boxes are exempt from splitting due to the balance condition.

4.1.1 Generating the Octree

The octree generation begins with the embedding of the root box set to a size that is a constant multiplied a bounding cube of the scene. Boxes are then selectively split and duplicated. The objective of this process is to make the intersection of P with the embedding of any box as simple as possible, such that it can be easily tetrahedrised. However, unnecessary splitting should be avoided as it would increase the number of tetrahedra produced. Octree generation consists of three main phases: the vertex phase, the edge phase and the facet phase.

The Vertex Phase: *A vertex cone of a box b is defined as a set of P faces $F_1, F_2, ..., F_k$ that satisfy the following:*

1. The set consists of only one vertex and all of its superfaces. The vertex itself is known as the apex of the vertex cone.
2. The vertex is contained in b.

3. The faces $F_1, F_2, ..., F_k$ are exactly the faces incident upon P?ex(b).

A box is defined to be vertex crowded if the following conditions are true:

1. There is a P vertex v in b.
2. The superfaces of v are not the only faces of P incident upon P?ex(b).

Equivalently, a box is vertex crowded if it contains a vertex that is not the apex of a vertex cone.

Boxes that are vertex crowded are split recursively, with the balance condition being propagated after each split, until no boxes that are vertex crowded remain. The process will terminate when the box size becomes a constant factor smaller than the distance between a given vertex and the P face that is not a superface of that vertex. Thus, when determining whether or not a box b is to be split, it is necessary to know which P faces bound P?ex(b).

Vertex Centring: Vertex centring is a one-time reorganisation of the boxes that is intended to increase the distance from a given vertex to the boundary of the box that contains it. Boxes whose embeddings contain a P vertex will be called vertex boxes. It can be shown (S. Mitchell, S. Vavasis [16]) that after the vertex phase, every box that is balance adjacent to a vertex box is either equal in size to the vertex box, or double its size. The vertex-centring step proceeds as follows for every vertex box b containing *vertex v:*

1. Split the boxes balance adjacent to b that are twice its size.
2. Merge b with the seven other boxes that share the corner of b that is closest to v to form the vertex box B.

Note that after the first step, b is balance adjacent to 26 boxes of equal size and b and its balance adjacent boxes are arranged in a 3x3x3 group.

B is then marked as protected and will never be split again. The merging process may violate the balance condition but this factor is ignored because the aspect ratio of the tetrahedra that are generated will still be bounded.

The Edge Phase: The edge phase proceeds as if the protected vertex boxes do not exist. Thus the extended box of a box in this phase does not extend into a protected box and the vertices of P are ignored.

An edge cone of a box b is defined as a set of P faces F_1, F_2, F_3 that satisfy the following:

1. The set consists of one edge and its two superfaces. The edge itself is known as the apex of the edge cone.
2. The edge is contained in b.
3. The faces F_1, F_2, F_3 are exactly the faces incident upon P?ex(b).

A box is defined to be edge crowded if it contains an edge but that edge and its superfacets are not the only faces incident upon P?ex(b). Equivalently, a box is edge crowded if it contains an edge that is not the apex of an edge cone.

As in the vertex phase, boxes that are edge crowded are split recursively, with the balance condition being propagated after each split, until no boxes that are edge crowded remain. To determine whether or not a box is edge crowded, a list of P faces that bound P?ex(b) is required.

It is desirable to increase the distance from a given edge to the boundary of the box containing it. However, an analogue of vertex centring cannot be applied at this stage because the edge box may be balance adjacent to a vertex box. Instead every unprotected box containing an edge as well as its unprotected balance adjacent boxes is split. The balance condition is propagated and then the edge box and the boxes balance adjacent to it are marked as protected. The distance mentioned above has to be increased at a later stage in the algorithm by warping the boxes.

The Facet Phase: The facet phase proceeds as if the boxes that were protected during the vertex and edge phase do not exist. Thus the extended box of a box during this phase does not extend into a protected box and the edges and vertices of P are ignored. In order to maintain consistency in terminology, facet cones and the concept of facet crowded are defined.

An facet cone of a box b is defined as the single face F_1 that satisfies the following:

1. F_1 is contained in b.
2. F_1 is the only face incident upon P?ex(b).

A box is defined to be facet crowded if it contains a facet but that facet is not the only face incident upon P?ex(b). Equivalently, a box is facet crowded if it contains a facet that is not the apex of a facet cone.

As in the edge and vertex phases, boxes that are facet crowded are recursively split until no facet crowded boxes remain. The balance condition is maintained after each split.

The facet boxes and the boxes balance adjacent to facet boxes are split, with the balance condition being maintained. For consistency facet boxes and boxes balance adjacent to them are protected, although the octree generation is complete after this phase and thus no more splitting will occur.

4.1.2 Triangulation

At this point there is deviation from the approach of Mitchel and Vavasis [16]. Due to implementation time constraints, the algorithm was shortened, thus the program proceeds directly to tetrahedrisation of empty boxes. Thus the tetrahedrisation no longer conforms to the object boundaries. There is now a space between object boundaries and their nearest tetrahedra. This space corresponds to boxes containing the boundary components. However, no object boundaries are intersected. Only leaf boxes need to be considered from this point onwards. Note that if a box is adjacent to boxes that are smaller, faces of the large box are replaced with the faces of the smaller box.

Two Dimensional Triangulation: Each facet of every empty box is triangulated by adding a central vertex. This vertex is then joined to every segment along the boundary of the facet.

Three Dimensional Triangulation: The centroid of each empty box is found. Tetrahedra are formed by taking the convex hull of the centroid with each of the surface triangles.

4.1.3 File Output

The tertahedra produced in the previous stage are written out to file, using indexed lists of vertices, edges and faces to represent the tetrahedra. The indices of the neighbours of each tetrahedra as well as the face through which that neighbour can be reached are also stored for each tetrahedron.

4.2 From-point Visibility

The from-point visibility algorithm uses a recursive method to traverse the tetrahedrisation. This will occur until a face of a mesh is encountered, the edge of the tetrahedrisation is reached, or a face is determined not to be projected through.

4.2.1 N-Sided Polygon Clipping

During each projection into a tetrahedron, a method is needed for determining if a projected polygon intersects a face. The clipping of polygons is such a technique, and due to the nature of the problem, a method is needed for dealing with an arbitrary *N*-sided clipping region. In order to achieve this, an approach similar to the Sutherland-Hodgeman clipping algorithm [24] is used.

The clipping planes are defined using the convention of the plane normal, and a point on the plane. A *frustum* is a collection of planes that a polygon will be clipped against. A polygon is simply an ordered list of points.

Any convex polygon can be used generate a view frustum. This is done, by walking the points of the polygon in order, using point pairs and the viewpoint to generate the plane normals (all planes in the viewing frustum will use the viewpoint as their 'point on plane'). This is done using the cross-product of the vectors from the viewpoint to both points.

Once the clipping planes are defined, in order to clip a polygon against the frustum, we simply clip the polygon against each plane making up the frustum. After the polygon has been clipped against a plane, this clipped polygon serves as input for the next clipping iteration. This is done until the polygon has been clipped against all planes in the frustum (it should be noted that the order of clipping does not matter).

4.2.2 Visibility Algorithm

The recursive method for visibility determination was implemented as follows:

Given a specified viewpoint, determine which tetrahedron you currently occupy (this is at worst case a linear search of the list of tetrahedra).

Once the occupied tetrahedron has been found, the algorithm projects recursively as follows:

For each face of the initial tetrahedron:

> If there is a neighbour present, then create a polygon representation of the face, and recursively call the function, passing the polygon, and viewpoint.
>
> If no neighbour is present, determine if it is part of a mesh. If so, mark it as visible.

When the function is called in a tetrahedron that is not the initial tetrahedron, the following takes place:

For each face of the tetrahedron:

> If the face doesn't return to the previous/calling tetrahedron:
> If a neighbour exists through the face, then create a frustum based upon the incoming polygon and viewpoint. It is then used it to clip against the face under consideration, and if it returns a polygon of at least 3 points, a recursive projection is made into the neighbouring tetrahedron, using the clipped polygon as input.
>
> If no neighbour exists:
> If face is part of a mesh, perform a clipping of the face against the incoming polygon. If the output contains at least 3 points, then the mesh is visible from the viewpoint, and should be marked as such.

This is done until recursion terminates, yielding a marking of all visible mesh faces.

Once this has been done, all that remains is to traverse the list of visible mesh faces and send them for rendering.

5 RESULTS

5.1 Tetrahedrisation

Figure 2 below shows the visualisation of an example test scene that has been loaded in, prior to tetrahedrisation.

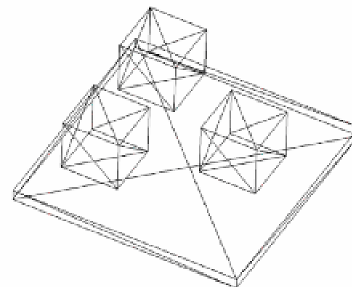

Figure 2: An example scene.

Figure 3 shows the octree generated for the test scene in figure 2, after the vertex phase but before vertex centring.

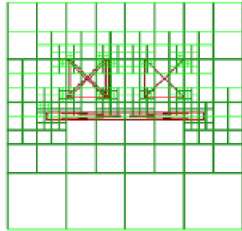

Figure 3: Side view of Octree generated for example scene.

Figure 4 shows the tetrahedrisation of the test scene in figure 2. The image on the left shows the same view as in figure 3. The image on the right shows the scene from a rotated view. There are an excessive number of tetrahedra, even for this simple scene.

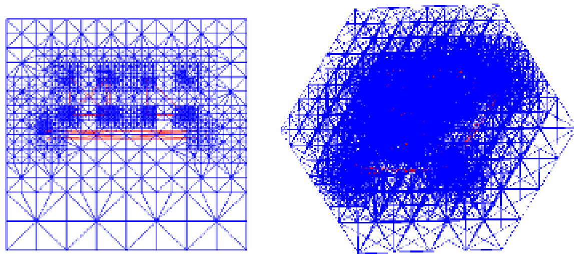

Figure 4: Tetrahedrisation of a test scene.

5.2 From-point Visibility

The from-point visibility algorithm was tested on a simple test scene (shown below in Figure 5). It comprises an 'I- shaped' region comprised of 30 tetrahedra. One side of the region was marked as an object mesh for testing purposes (shown in blue).

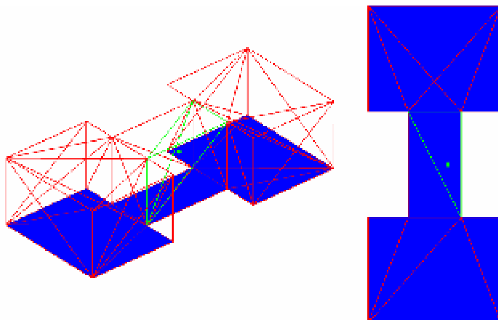

Figure 5: Simple test scene.

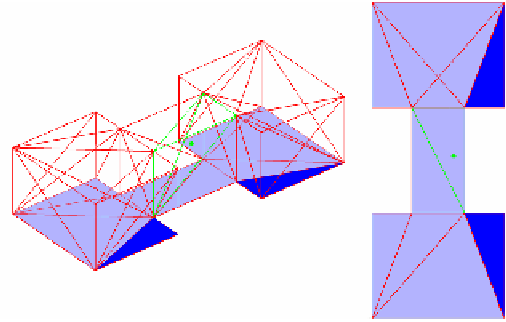

Figure 6: Visibility results.

After the visibility algorithm has been run, the visible object mesh faces are marked, and drawn in light blue (as shown in Figure 6).

The results demonstrate that the from-point algorithm is able to generate the correct visibility information using the tetrahedrisation.

6. CONCLUSIONS

Two requirements of the tetrahedrisation are not met. The tetrahedrisation does not conform to the object boundaries. Space is left between the tetrahedrisation and the object boundary. Due to this fact, although there is no space between neighbouring tetrahedra, the viewpoint space is not fully tetrahedrised. Considering desirable characteristics, the tetrahedra are of good quality. This is trivial however because the likelihood of poor quality tetrahedra is far greater where the tetrahedra would meet the object boundaries.

Despite the shortcomings of the implementation, it was sufficient to be used for testing the visibility culling algorithm as intended. In order to do this, the faces of tetrahedra that do not have neighbouring faces are marked as object faces.

The from-point visibility technique has achieved the desired result of being able to identify visible primitives in a scene. The tuning of the algorithm for conservative and aggressive performance is, however, still not implemented. It will be much easier to implement the aggressive performance within the current framework, as this requires only a testing of the angles between points on the polygon being projected.

The ability of the system to perform quickly for small scenes is encouraging, but the system performance will degrade as the number of tetrahedral increases. This however, still leaves open the possibility of using the approach as a pre-process, or during rendering of high quality images from 3-D scenes.

6 FUTURE WORK

6.1 Tetrahedrisation

There are many ways in which this projected can be expanded and improved. The main focus would be to fully complete the implementation of the design, which would remove many of the deficiencies of this specific implementation. Additionally, the suggested techniques to reduce the number of tetrahedra, at the expense of aspect ratio, could be implemented. It has been shown theoretically, that if the constraint of bounded aspect ratio is ignored, the number of tetrahedra produced is linear in the number of vertices (M. Bern, D.Eppstein, J. Gilbert [4]).

The following improvements could be made to the face in box tests: The cube – edge intersection test that is used is prone to floating point round-off errors, thus the edge may slip through the "crack" between adjacent cube faces. A more robust approach is suggested by A.Pateth [30] that involves testing whether the origin is contained in the convex solid obtained by sweeping a unit cube being centred at one edge endpoint to the other. A case analysis, such as that used by the three dimensional Cohen – Sutherland polygon clipping algorithm could also be incorporated to remove the need to perform any such testing for many cases.

In order to reduce the memory requirements of the program, at the expense of performance, less information could be stored for each box of the octree. For example, the coordinates of each box as well as the pointers to the faces its embedding contains could be calculated when required as the octree is traversed.

Over and above the techniques that were to be considered in this project, there are further techniques to reduce the number of tetrahedra. Heuristics can be used to achieve significant reductions. M. Bern et al [4] provide a convincing graphical example of their effectiveness in two dimensions.

6.2 From-Point Visibility

Shadows and Lighting

The system currently generates projections of polygons onto visible surfaces in the scene. In future, it could be advantageous to use these to generate more realistic lighting and shadow effects in the scenes.

Space division techniques for improved searching of tetrahedra

Techniques such as the BSP-Tree could be used to improve search times for finding the tetrahedron currently occupied by the viewpoint.

AI-Agent Vision

The visibility information generated for the purposes of rendering could instead be used to provide an artificial agent with information about the artificial/modelled world in which it is present.

REFERENCES

[1] K. Bala, J. Dorsey, and S. Teller. *Ray-traced Interactive scene editing using ray segment trees. Eurographics Rendering Workshop*, June 1999. Held in Granada, Spain.

[2] D. Bartz, M. Meiner, and T. Httner. OpenGL-assisted occlusion culling for large polygonal models. *Computer & Graphics*, 23(5):667-679, 1999.

[3] M. Bern and Paul Plassmann. "Mesh Generation", *Chapter 6 in Handbook of Computational Geometry, J.-R. Sack and J. Urrutia, eds., Elsevier Science*, 1999.

[4] M. Bern, D. Eppstein. "Mesh Generation And Optimal Triangulation". *Computing in Euclidean Geometry, Edited by Ding-Zhu Du and Frank Hwang, World Scientific, Lecture Notes Series on Computing -- Vol. 1,* 1995.

[5] M. Bern, D.Eppstein, J. Gilbert. "Provably Good Mesh Generation" *IEEE Symposium on Foundations of Computer Science,* 1990.

[6] J. Bittner, V. Havran, and P. Slavik. Hierarchical visibility culling with occlusion trees. In Proc. of Computer Graphics International, pgs 207-219, June 1998.

[7] P. Cavalcanti, U. Mello. "Three-dimensional Constrained Delaunay Triangulation: A Minimalist Approach". *Proceedings* 8th *International Meshing Roundtable,* 1999.

[8] S. Cheng, K. Dey, H. Edelsbrunner, M. Facello, S. Teng, "Sliver Exudation": *Symposium on Computational Geometry,* 1999.

[9]D. Cohen-Or, Y. Chrysanthou, C. T. Silva and F. Durand. A Survey of Visibility for Walkthrough Applications. *Course 30, SIGGRAPH*, August 2001.

[10] S. Coorg and S. Teller. Real-time occlusion culling for models with large occluders. *1997 Symposium on Interactive Computer Graphics*, pages 83-90, April 1997.

[11] Q. Du, D. Wang. "Constrained Boundary Recovery For Three Dimensional Delaunay Triangulations". *International Journal For Numerical Methods In Engineering,* 2004.

[12] D. Eppstein, J. Sullivan, A. Üngör. "Tiling Space And Slabs With Acute Tetrahedra: "*Computational Geometry: Theory and Applications, Volume 27 Issue 3,* 2003."

[13] N. Greene, M. Kass, and G. Miller. Hierarchical z-buffer visibility. *Proceedings of SIGGRAPH 93*, pages 231-240, 1993.

[14] T. Hudson, D. Manocha, J. Cohen, M. Lin, K. Hoff, and H. Zhang. Accelerated occlusion culling using shadow frustra. *In Proc. 13th Annual ACM Symposium on Computational Geometry*, pgs 1-10, 1997.

[15] J. T. Klosowski, and C. T. Silva. The prioritized-layered projection algorithm for visible set estimation. *IEEE Transactions on Visualisation and Computer Graphics*, 6(2):108-123, April-June 2000, ISSN 1077-2626.

[16] S. Mitchell, S. Vavasis."Quality Mesh Generation In ThreeDimensions". *Symposium on Computational Geometry,* 1992.

[17] S. Morien. ATI Radeon Hyper-Z technology. *In presentation at Hot3D Proceedings, part of Graphics Hardware Workshop*, 2000.

[18] S. Nirenstein. *Fast and Accurate Visibility Pre-processing. PhD thesis, University of Cape Town*, 2003

[19] S. Owen. "A Survey Of Unstructured Mesh Generation Technology". *Proceedings 7th International Meshing Roundtable,* 1998.

[20] V.Rajan. "Optimality Of The Delaunay Triangulation in R^d". *Proceedings 7th ACM Symposium Of Computational Geometry,* 1991.

[21] J. Ruppert (1995). "A Delaunay Refinement Algorithm For Quality 2-Dimensional Mesh Generation". *J. Algorithms,* 1995.

[22] E. Schönhardt: "Über Die Zerlegung Von Dreieckspolyedern in Tetraeder". *Math Annalen*, 1928.

[23] I. E. Sutherland, R. F. Sproull, and R. A. Schumaker. A characterisation of ten hidden surface algorithms. *ACM Computer Surveys, 6(1):1-55*, March 1974.

[24] I. E. Sutherland, G. W. Hodgeman, "Reentrant Polygon Clipping," *Communications of the ACM*, 17, 32-42, 1974.

[25] K. Weiler, P. Atherton. *Hidden surface removal using polygon area clipping. ACM SIGGRAPH Computer Graphics, Proceedings of the 4th Annual conference on Computer Graphics and iterative techniques*, Volume 11, Issue 2, July 1977.

[26] H. Zhang, D. Manocha, T. Hudson, and K. E. Hoff. Visibility culling using hierarchical occlusion maps. *In Turner Whitted, editor, SIGGRAPH 97 Conference Proceeding, Annual Conference Series,* pgs 77-88. ACM SIGGRAPH, Addison Wesley, August 1997.

This document was created with Win2PDF available at http://www.daneprairie.com.
The unregistered version of Win2PDF is for evaluation or non-commercial use only.